0508 US/P

Docket No.: GR 98 P 8110

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applic. No. : 09/816,933
Inventor : Christian Siemers
Filed : March 23, 2001
Title : Program-Controlled Unit

DECLARATION under 37 C.F.R. § 1.131

The undersigned hereby declares:

The invention of the above-identified application was "reduced to practice" at least as early as one day prior to September 14, 1998.

The undersigned inventor personally wrote and then submitted on January 22, 1998 to Siemens AG (attention Mr. Hassa), assignee of the subject invention, an invention disclosure entitled "Universal configurable blocks - a novel microarchitecture for field programmable logic devices (FPL)." The aforesaid invention disclosure described the invention as it was later disclosed and claimed in the above-identified patent application.

Enclosed herewith, as corroborating evidence, are a letter dated January 22, 1998 signed by the inventor Mr. Siemers with an attachment in the form of an invention disclosure entitled "Universal configurable blocks - a novel microarchitecture for field programmable logic devices (FPL)" which was submitted initially to Siemens AG and subsequently to the patent firm of Jannig & Repkow, which substantiate that the undersigned inventor invented and "reduced to practice" the claimed invention of the instant patent application at least one day prior to September 14, 1998.

The undersigned declares that all statements made herein of his own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001 and such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Christian Siemers

Christian Siemers

Date: Dec. 5th, 2005.

Erf. Nr. 326356

26. JAN 1998

Dr. Christian Siemers
St. Godehard Straße 18
31139 Hildesheim
Tel. 05121/267775

ZT GG VM Mch M
Eing. 26. Jan. 1998
GR
Frist

22.01.1998

Dr. Christian Siemers, St. Godehardstraße 18, 31139 Hildesheim

98E 8084
Erfindungsbeschreibg.

An
Siemens AG
ZT PA 6
Z.Hd. Hrn. Oliver Hassa
Postfach 22 16 34

80506 München

HSS

Folgepatentanmeldung

Sehr geehrter Hr. Hassa,

zunächst möchte ich mich einmal bei Ihnen für Ihre Weihnachts- und Neujahrswünsche sehr herzlich bedanken, verbunden mit den besten Wünschen für Sie in 1998. Zugleich kann ich Ihnen versichern, daß ich mich um die Erhaltung meiner Aktivitäten bemühen werde.

Anläßlich meines letzten München-Besuchs am 21.01.98 habe ich bei Hrn. Platzöder den Vertrag für die beiden Patentvorschläge aus 1997 unterzeichnet, so daß dies nunmehr auch vollständig geklärt ist. Bei dem anschließenden Gespräch bei Hrn. Dr. von Wendorff sind wir beide jedoch übereingekommen, daß eine Anmeldung in einem verwandten Gebiet noch dringender ist als in den nunmehr übertragenen.

Es handelt sich dabei um eine neue Architektur für Feldprogrammierbare Logikbausteine (FPL), die mit einem Teil aus dem >S<puter eng verknüpft ist. Diese Architektur heißt UCB (Universal Configurable Block) und wird nach unserer gemeinsamen Überzeugung in naher Zukunft wirklich zum Tragen kommen, da diese Plattform gemeinsam für FPL und Mikrocontroller Verwendung finden kann.

Ich habe daher die notwendigen Seiten aus dem Gesamtkonzept herausgelöst und ergänzt und übersende Ihnen hiermit dieses Architekturkonzept zur Prüfung, ob dieses für eine Patentanmeldung Verwendung finden könnte. Ich bitte um Entschuldigung, daß ich Sie hiermit quasi überfalle, aber in diesem Fall halte ich Eile für geboten.

Ich möchte Sie bitten, möglichst umgehend mit mir Kontakt aufzunehmen. Sie erreichen mich noch diese Woche unter den Rufnummern 0481/8555-832 (dienstlich), 05121/267775 (Donnerstag und Freitag) sowie meiner Funkrufnummer 0177/2981003.

Mit freundlichen Grüßen

Chr. Siemers

Anlage

98 E 8084 - Erfindungsbeschreibung.

Prof. Dr. Christian Siemers, Hildesheim:

Universal Configurable Blocks – eine neuartige Mikroarchitektur für Field Programmable Logic Devices (FPL)

1 Einleitung

Rekonfigurierbare bzw. strukturierbare Hardware, häufig als Feldprogrammierbare Logikbausteine (Field Programmable Logic Devices, FPL) bezeichnet, ist in der Bekanntheit deutlich gestiegen: War die strukturierbare Hardware vor einiger Zeit etwas für Spezialisten, die sich mit dem Ersatz von 'fertigen' Bausteinen befaßten und sogenannte Glue Logic bzw. State Machines einfacher bis mittlerer Komplexität darin implementierten, so hat sich die Gruppe der Forscher und Entwickler in Richtung Informatik erweitert. Die Gründe für diesen neuerlichen Attraktivitätsgewinn liegen in der wachsenden Größe von feldprogrammierbaren Bausteinen, die nunmehr zur Implementierung ganzer Systeme bzw. Algorithmen in Hardware geeignet sind, bei gleichzeitig sinkenden Preisen.

Damit hat sich auch der Charakter von Hardware-Applikationen deutlich gewandelt. Rekonfigurierbare Hardware ist sehr schnell, im Vergleich zur Software ähnlich programmierbar, wenn auch im Detail deutliche Unterschiede bestehen. Es ist jedoch das Denken in ganzen Algorithmen bzw. in wesentlichen Kernteilen, das die Einbettung von FPLs in ein Rechnersystem nunmehr beeinflußt. Um hier einige Beispiele zu nennen: DSP-Algorithmen, sehr schnelle Steuerungssysteme, Hardware-Beschleuniger [6]. Die Integration von Prozessoren und FPL wird im allgemeinen als ein sehr wesentlicher Teil des Hardware/ Software Co-Designs [1][6] angesehen.

Mit diesem hier beschriebenen Ansatz wird nunmehr eine FPL-Architektur vorgestellt, die sich aufgrund ihrer Struktur wesentlich besser in ein Mikroprozessor/-controllersystem integrieren läßt, da sich der Ablauf von 'Software' im Prozessor und im FPL auf eine gemeinsame Basis stellen läßt. Hierzu muß natürlich der Begriff der Software näher beschrieben und eingeschränkt werden. Software für einen Mikroprozessor/Mikrocontroller besteht auf der untersten Ebene aus Instruktionen, die gemäß dem von-Neumann-Modell sequentiell geladen und abgearbeitet werden. Dies wird für superskalare Prozessoren, die eine nebenläufige Ausführung gestatten, dahingehend variiert, daß die Ergebnisse denen eines sequentiellen Ablaufs entsprechen (Ergebnissequentialität).

Diese Form der Software auf unterster Ebene wird zumeist von optimierenden Compilern, seltener durch eine direkte Assemblerprogrammierung erzeugt. Ein sequentieller Fluß von Instruktionen bewirkt Aktionen in einer dafür geeigneten Ablaufeinheit, in der Regel ein Prozessor, so daß man dies als Kontrollfluß-dominiert bzw. prozedural (*control flow procedural*) bezeichnet. Im Unterschied hierzu werden FPLs in ihren Strukturen konfiguriert, diese strukturale Programmierung steuert einen Datenfluß. Ziel dieses Papers ist es, eine FPL-Struktur zu definieren, die mit Hilfe des bereits für den >S<puter [2] angegebenen Algorithmus zur Umsetzung von sequentiellen Instruktionen in strukturale Informationen programmiert werden kann und somit in der Lage ist, diesen sequentiellen Instruktionsstrom ohne weitere Steuereinheiten auszuführen.

Diese Form der Programmierung wird als *struktur-prozedurale Programmierung* (*Procedural Driven Structural Programming, PDSP*) bezeichnet.

Die in diesem Paper vorgeschlagene neue Architektur für Feldprogrammierbare Bausteine kann demnach zugleich als eine Stand-Alone-Einheit betrachtet werden, die durch eine zumeist kurze Sequenz von Instruktionen programmiert wird und dieses Programm selbständig ausführt, sie kann gleichzeitg als Ausführungseinheit in Prozessoren Verwendung finden. Dies steht in engem Zusammenhang mit dem >S<puter-Konzept [2][3][4], bei dem diese Architektur in nur leicht variierter Form bereits Einzug gehalten hat. Die neuartige FPL-Architektur, mit Universal Configurable Block (UCB) bezeichnet, wird insbesondere für Systeme vorgeschlagen, die im Rahmen eines Hardware/Software Co-Designs zusammen mit Prozessoren Teile der Gesamtapplikation in sich aufnehmen sollen.

2 Universal Configurable Blocks (UCB)

Die in [2][3][4] eingeführten Functional Units innerhalb des >S<puters wurden zur Anpassung einer Struktur an einen sequentiellen Instruktionsfluß genutzt – quasi als Speichermedium für eine Makroinstruktion, die einem Hyperblock entspricht. Diese Sichtweise wird durch die Angabe eines Algorithmus für eine Umrechnung des Instruktionsflusses in die Struktur unterstützt.

Die Functional Units lassen sich jedoch aus dem Kontext der s-Unit herauslösen und gesondert betrachten. Dies erfolgt hier mit Hinblick auf die Allgemeingültigkeit der nachfolgenden Überlegungen. Eine derartige Struktur, wie sie in den Functional Units vorhanden ist, kann als neue Struktur für feldprogrammierbare Bausteine betrachtet werden. Der Algorithmus zur Bestimmung der Struktur aus den Instruktionen schafft dann ein Interface zwischen der Software und der Struktur, das auch im Rahmen des Hardware/Software Co-Design sehr wichtig ist.

Zunächst werden jedoch die Functional Units zu den eigenständigen Universal Configurable Blocks (UCBs) erweitert.

2.1 Die Definition der Zielhardware: Universal Configurable Block

Der grundlegende Aufbau der Zielhardware wurde bereits als Functional Unit innerhalb der >S<puter-Architektur vorgestellt ([2] [3] [4]). Hierzu wurde die ALU-Struktur in kleinere Einheiten, die mit AUs und CoUs (Arithmetic Units, Compare Units) bezeichnet wurden, aufgelöst und durch ein Netzwerk konfigurierbarer Multiplexer und Demultiplexer miteinander verbunden. Abb. 2-1 zeigt das Aufbauprinzip, die im Rahmen dieses Kapitels nunmehr zu Universal Configurable Blocks (UCB) erweitert werden.

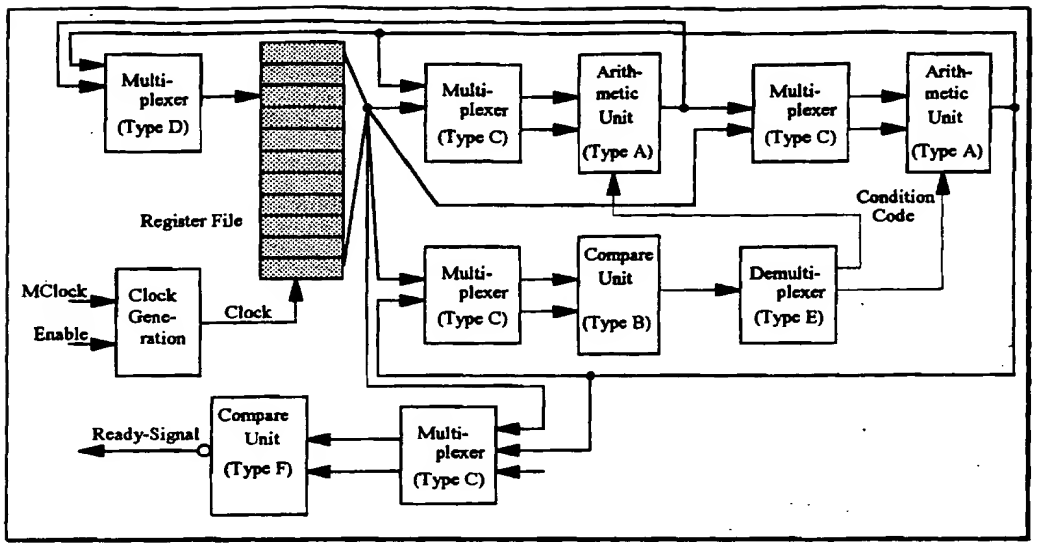

Abb. 2-1: Prinzipstruktur der UCBs

Ein derartiger UCB besteht demnach aus einem Registerblock als dem synchronisierenden Teil und einem konfigurierbaren, asynchronen Schaltnetz zwischen den Aus- und Eingängen des Registerblocks. Schreibzugriffe im Registerblock sind grundsätzlich getaktet. Der UCB entspricht in seinem Aufbau einer um die Taktgenerierung und die Generierung des Ready-Signals erweiterten Functional Unit mit folgenden, asynchron miteinander gekoppelten Bestandteilen:

- Arithmetic Unit (AU, Type A)
- Compare Unit (CoU, Type B)
- Multiplexer (Mul_C, Type C)
- Multiplexer (Mul_D, Type D)
- Demultiplexer (Demul, Type E)
- Compare Unit (CoU2, Type F): Diese gegenüber der in [2][3][4] vorgestellten Functional Unit neue Teileinheit kann das Ready-Signal für eine steuernde Einheit generieren. Hierzu werden wie für Typ B die Quellen per Multiplexer ausgewählt, zusätzlich kann man FALSE (Standardeinstellung) oder TRUE auswählen, um einen einzigen bzw. einen dauernden Durchlauf des Hyperblocks zu ermöglichen. Die Auswahl eines berechneten Ausgangssignals dieser Compare Unit hingegen ermöglicht die einfache Integration eines Schleifenzählers in Hardware. Die CoU2 wird im wesentlichen aufgebaut sein wie die CoUs (Type B). Die singuläre Verwendung für die Loop-Behandlung garantiert die Verfügbarkeit für ein explizites Schleifenende.

  CoU2-Teileinheiten können mehrfach in dem UCB vorhanden sein und einem erweiterten Interface zur Außenwelt dienen. Dies bedeutet, daß UCBs eigenständig mit anderer Hardware, beispielsweise Peripherieeinheiten in einem Mikrocontroller oder anderen UCBs interagieren können.
- Die Clock Generation Unit (CGU) erzeugt einen Takt für die Übernahme von Ergebnissen in Register. Sie koppelt im einfachsten Fall einen Mastertakt mit dem Enable-Signal einer

vorgeschalteten Einheit, beispielsweise eines Dispatchers oder eines Interface zum Speicher oder zu peripheren I/O Komponenten etwa via Load/Store-Pipeline.

Das Neuartige der UCBs als FPL-Architektur besteht also in folgenden Merkmalen:

- Die datenverknüpfenden Einheiten integrieren höhere Komplexitäten verglichen mit bisherigen FPL-Architekturen. Dies bedeutet die zumindestens partielle Abkehr von DNF- (Disjunktive Normalform) oder Look-Up-Table basierten, *logischen* Verknpüfungen hin zu *arithmetisch-logischen* Verknüpfungen, von denen jeweils eine, ggf. mehrere konfigurierbare in einer *Arithmetical Unit* (AU) vorhanden ist.
- Einhergehend mit dem vorgenannten Punkt ist die Blockung in den UCBs gröber als in den FPGAs (Field Programmable Gate Arrays), d.h. die Größe eines Blocks, der insgesamt für eine Makrooperation programmiert werden kann, ist größer und vergleichbar eher mit der von CPLDs (Complex Programmable Logic Devices). Dies bedeutet eine etwas geringere Flexibilität in der Anpassung der wirklich genutzten Ressourcen an die Anforderung, ergibt andererseits jedoch eine optimale Anpassung an Algorithmen, die in Mikrocontrollern ablaufen.
- Die Kopplung an Speicher und Input/Output-Elemente erfolgt über Load/Store-Pipelines sowie die Clock Generation Unit, um hier eine möglichst große Flexibilität zu erreichen. In handelsüblichen FPLs ist dies in sogenannten IO-Blöcken ohne größere eigenständige Funktionalität integriert, so daß ein Außenzugriff durch die strukturale Hardware intern gesteuert werden muß.
- Die Registerarchitektur ist besonders hervorzuheben, da die Register das synchronisierende Element innerhalb darstellen und zugleich mit der Außenwelt kommunizieren. Verknüpfungen zwischen zwei Register erfolgen der Architektur entsprechend asynchron.

2.2 Die UCB-Architektur im Verhältnis zur Prozessorarchitektur

Die Architektur des UCBs ist derart gewählt, daß fünf Gruppen von Instruktionen, wie sie in dem Befehlsstrom eines (superskalaren) Prozessors vorkommen, hierin ausführbar sind:

1. *Unkonditionierte Befehle* zur Bearbeitung von Daten einschließlich einer Datenkopie zwischen Registern: Diese Befehlsgruppe, im folgenden als 'normale Befehle' bezeichnet, beinhaltet arithmetische und logische Verknüpfung zwischen Daten zu neuen Werten sowie die Move-Befehle zur Kopie von Registerinhalten. Allgemeines Format dieser Befehle lautet

   <mnemonic> <destination reg.>, <source reg. 1>, <source reg. 2>

2. *Konditionierte Befehle* zur Bearbeitung von Daten bei Vorliegen einer Kondition: Diese Befehlsgruppe, im folgenden als 'bedingte Befehle' bezeichnet, beinhaltet grundsätzlich die gleichen Befehle wie unter 1. genannt, wobei in einer konkreten Realisierung natürlich nicht alle Verknüpfungen implementiert seien müssen. Das allgemeine Format lautet

   <mnemonic>p <destination reg.>, <source reg. 1>, <source reg. 2> <p-flag>,

   wobei durch das 'p' am Ende des Mnemonic die Abhängigkeit von der Bedingung ('predicated instructions') und durch das p-flag die Bedingung formuliert wird. Es wird

grundsätzlich angenommen, daß die Bedingung in Form eines Flags, bei Erweiterung des Modells auch durch logische Kombination mehrerer Flags definierbar ist.

3. Die *Predicate-Befehle* zur Bestimmung des Werts eines Bedingunsflags: Diese Befehlsgruppe, im folgenden durch 'pxx-Befehle' abgekürzt, bestimmt den Wert eines Bedingungsflags zur Laufzeit durch einen Vergleich, bei dem xx durch die Abkürzungen gt (Greater Than), ge (Greater than or Equal), eq (EQual), ne (Not Equal), le (Less than or Equal) und lt (Less Than) ersetzt wird, wobei andere Vergleiche ebenfalls integrierbar sind. Befehle dieser Gruppe besitzen somit das Format

   <mnemonic> <source reg. 1>, <source reg. 2>, <p-flag> .

   Sie sind mit den üblichen Branch-Befehlen vergleichbar und dienen deren Ersatz durch Anwendung der 'if-conversion' [5].

4. Die Loop-Befehle zur Schleifenwiederholung am Ende eines Hyperblocks: Diese Befehle, im Format

   loopxx <source reg. 1>, <source reg. 2>

   mit xx als Abkürzung der üblichen Vergleiche (ne, eq ...) codiert, führen die Verzweigung an den Anfang eines Hyperblocks durch, falls die Bedingungen wahr ist. Dementsprechend soll das Ready-Signal generiert werden, wenn die Schleifenbedingung nicht mehr erfüllt ist.

5. Die Intxx-Befehle zur Erzeugung von Signalen an die Außenwelt des UCBs: Diese Befehle, im Format

   intxx <source reg. 1>, <source reg. 2>, <int_signal>

   mit xx als Abkürzung der üblichen Vergleiche (ne, eq ...) einschließlich TRUE und FALSE, erzeugen im Programmfluß ein rückgeführtes Signal zum Interface des UCBs, falls die Bedingung wahr ist. Dieses Signal, in int_signal angegeben, kann zur Kommunikation mit anderen Einheiten genutzt werden. Intxx-Befehle stellen eine Verallgemeinerung der loopxx-Befehle dar, indem nicht nur das ausgezeichnete Ready-Signal, sondern allgemeine Hardwaresignalleitungen gesetzt werden.

Ein UCB soll optimal so dimensioniert sein, daß er im allgemeinen einen Hyperblock in sich zur Ausführung aufnehmen kann. Hierzu wird für die weitere Ausführung ohne Beschränkung angenommen, daß die Busse zwischen den Multiplexern und den zu verbindenden Teilblöcken in den gewünschten Dimensionierungen vorhanden sind und durch einfache Konfigurierungsbits schaltbar sind. Für die nachfolgende Formulierung des Konfigurierungsalgorithmus' wird zusätzlich ein vollständiges Netzwerk angenommen, wobei sich hier bei der Beschränkung auf Teilnetze Einschränkungen für den nachfolgenden Algorithmus ergeben würden.

2.3 Ein Beispiel zur Integration in die UCB-Architektur

Abschließend zu den bisherigen Betrachtungen soll ein Beispiel aus dem I/O-Bereich dargestellt werden. Timer-gesteuert wird ein AD-Wandler, dessen Wandlungsbreite mit 8 Bit angenommen wird, gestartet und der Wert mit einer 12-Bit-Zählmarke zusammen gespeichert. Der AD-Wert wird dann auf Über- und Unterschreitung spezifizierter Grenzen überwacht, wobei dann in eine weitere Routine verzweigt wird. Die Routine bricht nach 2048 Messungen ab.

Diese Anwendung erzeugt bei einer reinen Softwarelösung einige Probleme: Da ein typischer AD-Wandler, der in einem Mikrocontroller integriert ist, nicht spontan das Ergebnis liefert, also als Flashwandler arbeitet, sondern eine Wandlungszeit im Bereich einiger Mikrosekunden besitzt, muß bei exakter Ausführung der Anwendungsspezifikation einer der folgenden Wege bestritten werden:

- Der Timer löst einen Interrupt aus, innerhalb dessen Serviceroutine die Wandlung gestartet wird. Die Timerserviceroutine wird beendet, der AD-Wandler löst bei Beendigung der Wandlung ebenfalls einen Interrupt Request aus. In der nunmehr folgenden zweiten Serviceroutine wird der AD-Wert ausgelesen und verarbeitet.
- Der Timer löst einen Interrupt aus, innerhalb dessen Serviceroutine sowohl die Wandlung gestartet, auf das Wandlungsende gewartet und die Wandlungswerte verarbeitet werden.

Die 2. Variante bietet sich bei im Vergleich zu Interruptlatenzzeiten kurzen Wandlungszeiten an, ansonsten wäre die erste Variante zu bevorzugen, da hier viel Wartezeit gespart wird. In der Praxis existiert jedoch ein dritter Weg, der im Prinzip nicht exakt den Vorgaben entspricht, jedoch im allgemeinen zulässig ist: Die Wandlung selbst wird in die Lesepausen verlegt. Der zu einem Timer-Interruptzeitpunkt gelesene Wert entspricht dann der Wandlung der vorhergehenden Periode, nach dem Lesen wird eine neue Wandlung für die nächste Periode gestartet. Im wesentlichen ist bei diesem Verfahren die Zeitverschiebung und die Ungültigkeit des ersten Werts zu beachten.

Für einen üblichen Mikrocontroller wird normalerweise das in der Praxis relevante Verfahren einer Interrupt-gesteuerten Routine realisiert, die die Wandlung in die Lesepausen verlegt. Die ersten Varianten würden zu einem erheblich gesteigertem Zeitbedarf führen. Es wird weiterhin angenommen, daß die Abfrage und die Verarbeitung in der Serviceroutine des Timerinterrupts erfolgen.

```
int *p_adc, adc_value, upper_limit, lower_limit, adc_ready;
int adc_array[4096];
...
void hardware_thread readADC()
{
    int x = 0;
    while( x < 4096 )
    {
        if( adc_ready == 1 )
        {
            adc_value = *p_adc;          // Access to AD converter
            if( adc_value > upper_limit  ||  adc_value < lower_limit )
                out_of_range();          // call to exception routine
            adc_array[x++] = x;          // index information
            adc_array[x++] = adc_value;  // and adc value are stored
        }
    }
}
```

Abb. 2-2: C-Sourcecode für ADC-Programm

Die Vermutung liegt nun nahe, daß für derartige Applikationsteile UCBs ebenfalls zur Unterstützung von peripheren Controllerelementen dienen können. Die Programmierung des UCB kann sowohl in strukturaler als auch in sequentieller Weise entsprechend in einer (Software-)Hochsprache erfolgen – einen vorhandenen Compiler vorausgesetzt, der die PDSP-

Übersetzung beherrscht. Abb. 2-2 gibt den Algorithmus zur Aufnahme von 2048 AD-Werten, Speicherung des jweiligen Wertes mit zusätzlicher Indexinformation und Vergleich auf Über- bzw. Unterschreitung von Grenzen in C an:

Ein für die UCB-Architektur optimierender Compiler könnte diesen Sourcecode unter Berücksichtigung der neuen Instruktionen in folgenden Assemblercode übersetzen:

```
     mov     r1, p_adc          ; address of ADC
     mov     r4, 0              ; variable x
     mov     r5, 1              ; variable x+1
     mov     r6, adc_array      ; address of array
     ld      r2, upper_limit    ;
     ld      r3, lower_limit    ;
L0:  ld      r0, (r1)           ; AD value is loaded
     intgt   r0, r2, i1         ; and compared for int
     intlt   r0, r3, i2         ; generation
     st      (r6+r4), r4        ; the storage
     st      (r6+r5), r0        ;
     add     r4, r4, 2          ;
     add     r5, r5, 2          ;
     looplt  r4, 4096, L1       ;
```

Register contents:	
r0:	AD value
r1:	&ADC
r2:	upper_limit
r3:	lower_limit
r4:	x
r5:	x+1
r6:	&adc_array

Abb. 2-3: Generierung des Assemblercode durch optimierenden Compiler

Die dargestellte Generierung wurde unter verschiedenen Annahmen durchgeführt, die im einzelnen erläutert werden müssen. Zunächst fehlt die im C-Code angegebene Bedingung, daß die Schleife nur beim Vorliegen des ADC_READY-Signals durchlaufen werden darf, im Assemblercode vollständig. Es wird hierzu angenommen, daß die Bedingung, die quasi eine Triggerung darstellt, vom Compiler in ein Enable-Signal für den Takt umgesetzt werden kann (siehe hierzu auch Abb. 2-1). Dieses Enable-Signal steuert den Übernahmetakt für den UCB und entspricht damit einem Trigger. Ist diese Annahme unzulässig, dann müssen alle Befehle von dem Signal ADC_READY abhängig gemacht werden, was prinzipiell möglich ist, jedoch sehr viele Vergleichsressourcen kostet.

Zudem sind bedingte Interruptaufrufe (intxx) in dem Assemblercode vorhanden, die in einem UCB auf die Signalgenerierung abgebildet werden können. Diese Signalgenerierung an eine vorgelagerte Stufe muß dort zwischengespeichert und verarbeitet werden, etwa durch einen Interruptaufruf an den eigentlichen Prozessor.

Unter diesen Voraussetzungen entsteht folgende Struktur in einem UCB:

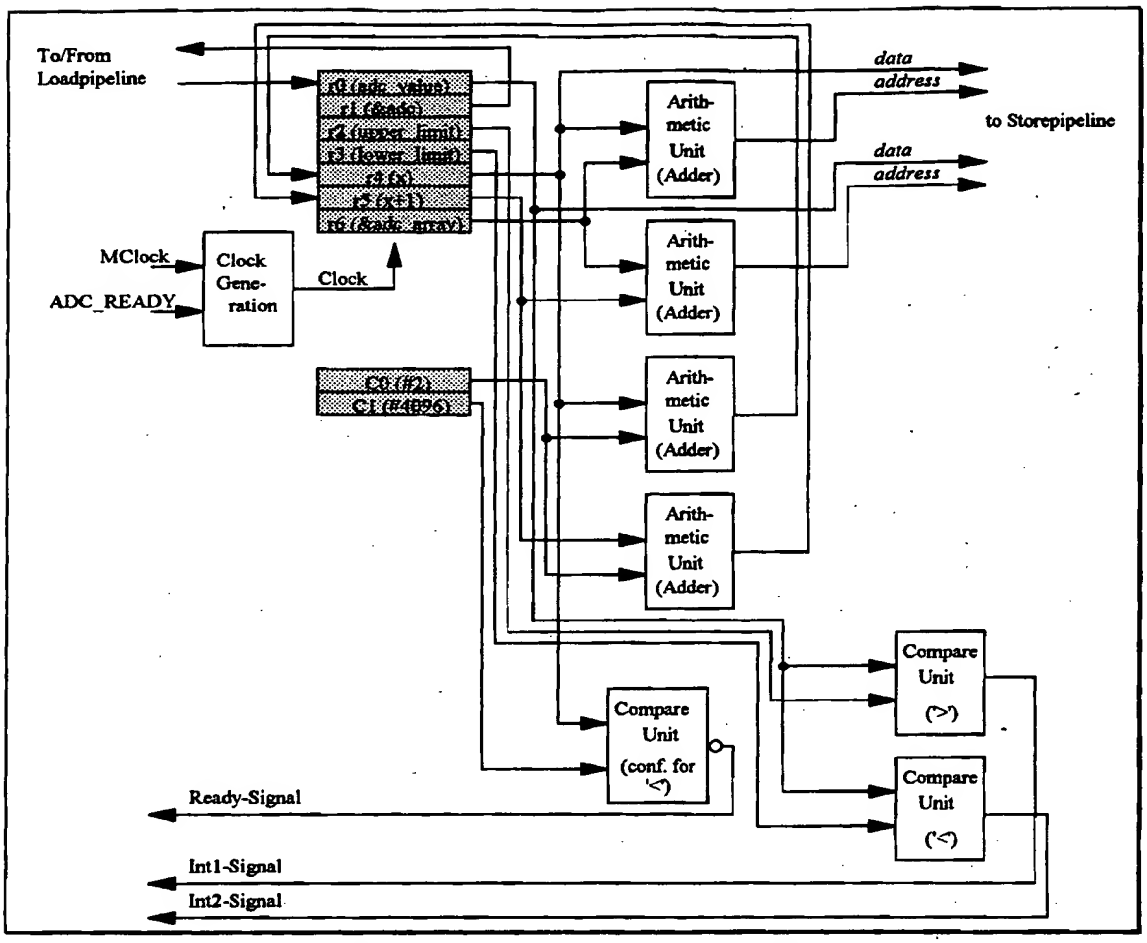

Abb. 2-4: Abbildung ADC-Routine in UCB

Dieses Beispiel zeigt die grundsätzliche Übertragbarkeit von Routinen für intelligente Peripherie in das Konzept der UCBs, wobei insbesondere für die Verwendbarkeit von prozeduralen Compilern einige Voraussetzungen gemacht werden mußten. Insbesondere muß der Systementwickler die Teile, die jeweils in einem UCB laufen sollen, als eigenständige Threads deklarieren (im C-Beispiel mit Hilfe des neuen Schlüsselwortes hardware_thread), während der Compiler selbst gefordert ist, dies für einen UCB zu übersetzen.

2.4 Zusammenfassung

Ziel dieses Konzepts war die Einführung einer neuen Architektur für feldprogrammierbare Logik, die zugleich in Mikrocontrollern bzw. -prozessoren integriert sein kann und mit Hilfe des bereits angegebenen PDSP-Algorithmus' (Procedural Driven Structural Programming) durch einen sequentiellen Instruktionsstrom ladbar ist. Diese Architektur zeichnet sich durch die Einführung einer neustrukturierten Hardware aus, die auf den Anwendungskreis optimiert ist.

Beispielhaft konnte hierbei gezeigt werden, in welcher Form die UCBs programmiert und betrieben werden können. Diese Struktur ist in der Lage, sowohl stand-alone als auch in Zusammenarbeit mit Mikrocontrollern betrieben werden zu können und zeigt durch die enge

Anpassung an die Architektur und Arbeitsweise von Prozessoren insbesondere im Rahmen eines Co-Designs sehr gute Ergebnisse.

3 Literatur

[1] De Micheli, G.; Gupta, R., „Hardware/Software Co-Design", Invited Paper in: Proceedings of the IEEE Vol. 85(3), Special Issue on Hardware/Software Co-Design, 341 .. 365, March 1997.

[2] Siemers, C.: Prozessor mit Pipelining-Aufbau. Anmeldung am 21.08.1996 unter der Nummer 196 34 031.4 beim Deutschen Patentamt.

[3] Siemers, C.; Möller, D.P.F., „Der >S<puter: Ein dynamisch rekonfigurierbares Mikroarchitekturmodell zur Erreichung des maximalen Instruktionsparallelitätsgrades", Vortragsband der 14. ITG/GI-Fachtagung Architektur von Rechensystemen ARCS '97, Rostock, September 1997, S. 133 .. 142. VDE Verlag, Berlin und Offenbach, 1997

[4] Siemers, C.; Möller, D.P.F., „The >S<puter: A Novel Microarchitecturemodel for the Execution of Instructions inside Processors", in: Xu De, K.-E. Großpietsch, Ch. Steigner (Eds.): Proceedings of the Second Sino-German Workshop on Advanced Parallel Processing Technologies APPT '97, Koblenz, September 1997, p. 75 .. 82. Fölbach Verlag, Koblenz, 1997.

[4] Smith, James E.; Sohi, Gurindar S., „The Microarchitecture of Superscalar Processors", Invited Paper in Proceedings of the IEEE, Special Issue on Microprocessors, Vol. 83 (12), p. 1609 .. 1624, 1995.

[5] Wen-Mei W. Hwu et. al., „Compiler Technology for Future Microprocessors", Invited Paper in Proceedings of the IEEE Vol. 83 (12) , Special Issue on Microprocessors, 1625 .. 1640, Dec. 1995.

[6] R.W. Hartenstein, J. Becker, R. Kress, „Custom Computing Machines vs. Hardware/Software Co-Design: From a Globalized Point of View", *6th Int. Workshop on Field Programmable Logic and Appl., FPL'96*, Darmstadt, Sept. 1996, Lecture Notes in Computer Science 1142, Springer 1996.

Docket No.: GR 989 P 8110
Application No. 09/816,933

CERTIFICATION

I, the below named translator, hereby declare that: my name and post office address are as stated below; that I am knowledgeable in the English and German languages, and that I believe that the attached text is a true and complete translation of a letter from Dr. Christian Siemers, dated January 22, 1998 to Mr. Hassa, including an invention disclosure (pages 2-12).

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Hollywood, Florida

Birgit Bartell

November 30, 2005

Lerner & Greenberg, P.A.
P.O. Box 2480
Hollywood, FL 33022-2480
Tel.: (954) 925-1100
Fax.: (954) 925-1101

Letter from Dr. Christian Siemers, Hildesheim, to Siemens AG, Munich, dated January 22, 1998

Follow-up patent application

Dear Mr. Hassa

Firstly I would like to thank you sincerely for your Christmas and New Year wishes, coupled with my best wishes for you in 1998. At the same time I can assure you that I will endeavor to maintain my activities.

During my last visit to Munich on January 21, 1998, with Mr. Platzöder I signed the contract for the two patent proposals from 1997, and so this is now also completely clarified. In the subsequent conversation with Dr. von Wendorff, however, we both agreed that an application in a related field is even more urgent than in those now transferred.

This concerns a new architecture for field programmable logic devices (FPL), which is closely linked with part of the >S<puter. This architecture is called UCB (Universal Configurable Block) and we are both convinced that it will really come to fruition in the near future since this platform can be used jointly for FPL and microcontrollers.

I have therefore taken the necessary pages from the overall concept and supplemented them and I am hereby sending you this architecture concept for you to examine whether this could be used for a patent application. Forgive me for virtually bombarding you with this, but I consider haste to be imperative in this case.

Please would you contact me as soon as possible. You can still reach me this week under the telephone numbers 0481/8555-832 (office), 05121/267775 (Thursday and Friday) and my pager number 0177/2981003.

Yours sincerely
[signature]

Enclosure

Prof. Christian Siemers, Hildesheim:

Universal configurable blocks – a novel microarchitecture for field programmable logic devices (FPL)

1 Introduction

Awareness of reconfigurable or structurable hardware, often referred to as field programmable logic devices (FPL), has increased significantly among those working in the field: whereas a while ago structurable hardware was something for specialists who were concerned with the replacement of "finished" devices and implemented so-called glue logic or state machines of simple to medium complexity therein, the group of researchers and developers has expanded in the direction of information technology. The reasons for this recent gain in attractiveness can be found in the increasing size of field programmable devices, which are now suitable for implementing whole systems or algorithms in hardware, in conjunction with falling prices.

The character of hardware applications has thus also changed significantly. Reconfigurable hardware can be programmed very fast, similarly to software, even though there are significant differences in details. However, it is the thinking in whole algorithms or in essential core parts which now influences the embedding of FPLs in a computer system. To mention a few examples here: DSP algorithms, very fast control systems, hardware accelerators [6]. The integration of processors and FPL is generally regarded as a very important part of hardware/software co-design [1][6].

This approach described here now presents an FPL architecture which, on account of its structure, can be integrated significantly better into a microprocessor/microcontroller system, since the sequence of "software" in the processor and in the FPL can be set to a common basis. For this purpose, it is necessary, of course, for the term software to be more closely described and qualified. Software for a microprocessor/microcontroller consists at the lowest level of instructions which are sequentially loaded and processed in accordance with the von Neumann model. This is varied for superscalar processors, which permit concurrent execution, to the effect that the results correspond to those of a sequential sequence (result sequentiality).

This form of software at the lowest level is usually generated by optimizing compilers, less often by direct assembler programming. A sequential flow of instructions brings about actions in a sequence unit suitable therefor, generally a processor, and so this is referred to as control

flow dominated or *control flow procedural*. In contrast thereto, the structures of FPLs are configured and this structural programming controls a data flow. It is an aim of this paper to define an FPL structure which can be programmed with the aid of the algorithm already specified for the >S<puter [2] for converting sequential instructions into structural information and is thus able to execute this sequential instruction stream without further control units.

This form of programming is referred to as *procedural driven structural programming (PDSP)*.

The new architecture for field programmable devices which is proposed in this paper may accordingly at the same time be regarded as a standalone unit which is programmed by means of a usually short sequence of instructions and executes this program independently; it may simultaneously be used as an execution unit in processors. This is closely related to the >S<puter concept [2][3][4], in which this architecture in just a slightly varied form has already made an inroad. The novel FPL architecture, designated by universal configurable block (UCB), is proposed in particular for systems which, in the context of a hardware/software co-design, together with processors are intended to incorporate parts of the overall application in themselves.

2. Universal configurable blocks (UCB)

The functional units within the >S<puter that were introduced in [2][3][4] were used for adapting a structure to a sequential instruction flow; virtually as a storage medium for a macroinstruction corresponding to a hyperblock. This aspect is supported by the specification of an algorithm for converting the instruction flow into the structure.

However, the functional units can be detached from the context of the s-unit and be considered separately. This is done here with regard to the general validity of the considerations that follow. A structure of this type such as is present in the functional units can be regarded as a new structure for field programmable devices. The algorithm for determining the structure from the instructions then creates an interface between the software and the structure, which is also very important in the context of the hardware/software co-design.

Firstly, however, the functional units are extended to form the independent universal configurable blocks (UCBs).

2.1 Definition of the target hardware: universal configurable block

The fundamental construction of the target hardware has already been presented as a functional unit within the >S<puter architecture ([2][3][4]). For this purpose, the ALU structure was resolved into smaller units, designated by AUs and CoUs (arithmetic units, compare units), and connected to one another by a network of configurable multiplexers and demultiplexers. Fig. 2-1 shows the construction principle, which will now be extended to form universal configurable blocks (UCB) in the context of this chapter.

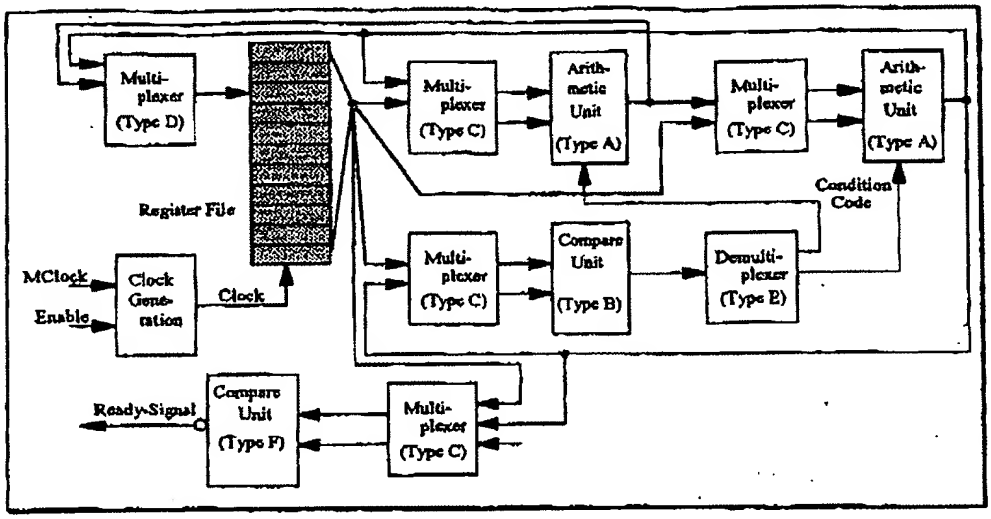

Fig. 2-1: Basic structure of the UCBs

A UCB of this type accordingly comprises a register block as the synchronizing part and a configurable, asynchronous switching network between the outputs and inputs of the register block. Write accesses in the register block are clocked, in principle. In terms of its construction, the UCB corresponds to a functional unit extended by the clock generation and the generation of the ready signal, with the following component parts that are asynchronously coupled to one another:

- Arithmetic Unit (AU, Type A)
- Compare Unit (CoU, Type B)
- Multiplexer (Mul_C, Type C)
- Multiplexer (Mul_D, Type D)
- Demultiplexer (Demul, Type E)

- Compare Unit (CoU2, Type F): This subunit, which is new in comparison with the functional unit presented in [2][3][4], can generate the ready signal for a controlling unit. For this purpose, as for Type B, the sources are selected by multiplexer; in addition, it is possible to select FALSE (standard setting) or TRUE in order to enable a single or a continuous pass of the hyperblock. The selection of a calculated output signal of this compare unit, by contrast, enables the simple integration of a loop counter in hardware. The CoU2 will essentially be constructed like the CoUs (Type B). The singular use for the loop treatment guarantees the availability for an explicit loop end.

  CoU2 subunits may be present multiply in the UCB and are used for an extended interface to the outside world. This means that UCBs can interact independently with other hardware, for example peripheral units in a microcontroller, or other UCBs.

- The clock generation unit (CGU) generates a clock for the acceptance of results in registers. In the simplest case, it couples a master clock to the enable signal of a unit connected upstream, for example of a dispatcher or an interface to the memory or to peripheral I/O components, for instance via load/store pipeline.

The novelty of the UCBs as FPL architecture thus consists in the following features:

- The data-combining units integrate higher complexities compared with previous FPL architectures. This means at least partially turning away from DNF (disjoint normal form) or look-up table based *logic* combinations toward *arithmetic-logic* combinations, of which there is in each case one configurable combination, if appropriate a plurality of configurable combinations, present in an *arithmetical unit* (AU).

- In association with the aforementioned point, the blocking in the UCBs is coarser than in the FPGAs (field programmable gate arrays), that is to say that the size of a block which can be programmed overall for a macro operation is larger and comparable rather with that of CPLDs (complex programmable logic devices). This means a somewhat lower degree of flexibility in the adaptation of the resources actually used to the requirement, but on the other hand produces an optimal adaptation to algorithms executed in microcontrollers.

- The coupling to memory and input/output elements is effected via load/store pipelines and also the clock generation unit, in order to achieve a maximum flexibility here. In commercially available FPLs, this is integrated in so-called IO blocks without a relatively

high degree of independent functionality, so that an external access has to be controlled internally by the structural hardware.

- The register architecture is to be particularly emphasized since the registers represent the synchronizing element within and at the same time communicate with the outside world. Combinations between two registers are effected asynchronously in accordance with the architecture.

2.2 The UCB architecture in relation to the processor architecture

The architecture of the UCB is chosen in such a way that five groups of instructions such as occur in the instruction stream of a (superscalar) processor can be executed therein:

1. *Unconditioned instructions* for processing data including a data copy between registers: This group of instructions referred to hereinafter as "normal instructions", comprises arithmetical and logic combination between data to form new values and also the move instructions for copying register contents. The general format of these instructions reads

   <mnemonic> <destination reg.>, <source reg. 1>, <source reg. 2>

2. *Conditioned instructions* for processing data when a condition is present: This group of instructions, referred to hereinafter as "conditional instructions", in principle comprises the same instructions as mentioned under 1., in which case, of course, not all combinations have to be implemented in a concrete realization. The general format reads

   <mnemonic>p <destination reg.>, <source reg. 1>, <source reg. 2> <p-flag>,

   the dependence on the condition ("predicated instructions") being formulated by the "p" at the end of the mnemonic and the condition being formulated by the p-flag. It is assumed in principle that the condition can be defined in the form of one flag, and in the case of the model being extended also by logic combination of a plurality of flags.

3. The *predicate instructions* for determining the value of a condition flag: This group of instructions, abbreviated hereinafter to "pxx instructions", determines the value of a condition flag at the execution time by a comparison where xx is replaced by the abbreviations gt (greater than), ge (greater than or equal), eq (equal), ne (not equal), le (less than or equal) and lt (less than), other comparisons likewise being able to be integrated. Instructions of this group thus have the format

<mnemonic> <source reg. 1>, <source reg. 2>, <p-flag>.

They are comparable with the customary branch instructions and serve to replace the latter by application of the "if-conversion" [5].

4. The loop instructions for loop repetition at the end of a hyperblock: These instructions, in the format

   loopxx <source reg. 1>, <source reg. 2>

   coded by xx as an abbreviation of the customary comparisons (ne, eq ...), perform the branching at the start of a hyperblock if the condition is true. Accordingly, the ready signal is to be generated if the loop condition is no longer fulfilled.

5. The intxx instructions for generating signals to the outside world with respect to the UCB: These instructions, in the format

   intxx <source reg. 1>, <source reg. 2>, <int_signal>

   with xx as an abbreviation of the customary comparisons (ne, eq ...) including TRUE and FALSE, generate in the program flow a fed-back signal to the interface of the UCB if the condition is true. This signal, specified in int_signal, can be used for communication with other units. Intxx instructions constitute a generalization of the loopxx instructions in that not only the distinguished ready signal but general hardware signal lines are set.

A UCB is to be optimally dimensioned such that it can generally incorporate a hyperblock in itself for execution. For this purpose, it is assumed for the further explanation without restriction that the buses are present between the multiplexers and the sub-blocks to be connected in the desired dimensionings and can be switched by simple configuration bits. A complete network is additionally assumed for the subsequent formulation of the configuration algorithm; restriction to sub-networks would in this case result in limitations for the subsequent algorithm.

2.3 An example of integration into the UCB architecture

An example from the I/O area will be illustrated to conclude the previous considerations. Under the control of a timer, an AD converter, the conversion width of which is assumed to be 8 bits, is started and the value is stored together with a 12-bit counting mark. The AD

value is then monitored with regard to exceeding and undershooting specified limits, the system then branching to a further routine. The routine terminates after 2048 measurements.

In the case of a pure software solution, this application generates a few problems: Since a typical AD converter integrated in a microcontroller does not supply the result spontaneously, that is to say operate as a flash converter, but rather has a conversion time in the region of a few microseconds, one of the following paths has to be taken for exact execution of the application specification:

- The timer triggers an interrupt, within the service routine of which the conversion is started. The timer service routine is ended; the AD converter likewise triggers an interrupt request at the end of the conversion. In the second service routine which then follows, the AD value is read out and processed.
- The timer triggers an interrupt, within the service routine of which the conversion is started, the end of conversion is awaited and also the conversion values are processed.

The 2nd variant is appropriate in the case of short conversion times in comparison with interrupt latencies; otherwise, the first variant would be preferable since much waiting time is saved in this case. A third path exists in practice, however, which in principle does not correspond exactly to the stipulations but is generally permissible: The conversion itself is moved into the read pauses. The value read at a timer interrupt instant then corresponds to the conversion of the preceding period and a new conversion for the next period is started after reading. In the case of this method, it is necessary essentially to take account of the time shift and the invalidity of the first value.

The practically relevant method of an interrupt-controlled routine which moves the conversion into the read pauses is normally realized for a customary microcontroller. The first variants would lead to a considerably increased time requirement. It is furthermore assumed that the interrogation and the processing are effected in the service routine of the timer interrupt.

```
int *p_adc, adc_value, upper_limit, lower_limit, adc_ready;
int adc_array[4096];
...
void hardware_thread readADC()
{
    int x = 0;
    while( x < 4096 )
    {
        if( adc_ready == 1 )
        {
            adc_value = *p_adc;          // Access to AD converter
            if( adc_value > upper_limit  ||  adc_value < lower_limit )
                out_of_range();          // call to exception routine
            adc_array[x++] = x;          // index information
            adc_array[x++] = adc_value;  // and adc value are stored
        }
    }
}
```

Fig. 2-2: C source code for ADC program

It seems a likely supposition that, for application parts of this type, UCBs can likewise serve for supporting peripheral controller elements. The UCB can be correspondingly programmed both in a structural manner and in a sequential manner in a (software) high-level language – presupposing that a compiler is present which controls the PDSP translation. Fig. 2-2 specifies the algorithm for taking up 2048 AD values, storing the respective value with additional index information and effecting comparison with regard to exceeding or undershooting limits in C:

A compiler which effects optimization for the UCB architecture could translate this source code taking account of the new instructions into the following assembler code:

```
     mov     r1, p_adc           ; address of ADC
     mov     r4, 0               ; variable x
     mov     r5, 1               ; variable x+1
     mov     r6, adc_array       ; address of array
     ld      r2, upper_limit     ;
     ld      r3, lower_limit     ;
L0:  ld      r0, (r1)            ; AD value is loaded
     intgt   r0, r2, i1          ; and compared for int
     intlt   r0, r3, i2          ; generation
     st      (r6+r4), r4         ; the storage
     st      (r6+r5), r0         ;
     add     r4, r4, 2           ;
     add     r5, r5, 2           ;
     looplt  r4, 4096, L1        ;
```

Register contents:	
r0:	AD value
r1:	&ADC
r2:	upper_limit
r3:	lower_limit
r4:	x
r5:	x+1
r6:	&adc_array

Fig. 2-3: Generation of the assembler code by compiler effecting optimization

The illustrated generation was carried out under various assumptions that must be specifically explained. Firstly, the condition specified in the C code that the system can run through the loop only when the ADC_READY signal is present is completely absent in the assembler code. It is assumed in this respect that the condition, which virtually constitutes a trigger, can be converted into an enable signal for the clock by the compiler (in this respect, also see fig. 2-1). This enable signal controls the acceptance clock for the UCB and thus corresponds to a trigger. If this assumption is impermissible, then all the instructions have to be made dependent on the ADC_READY signal, which is possible in principle, but costs very many comparison resources.

Moreover, there are conditional interrupt calls (intxx) present in the assembler code, which can be mapped onto the signal generation in a UCB. This signal generation to an upstream stage must be buffer-stored there and processed, for instance by means of an interrupt call to the actual processor.

Under these preconditions, the following structure arises in a UCB:

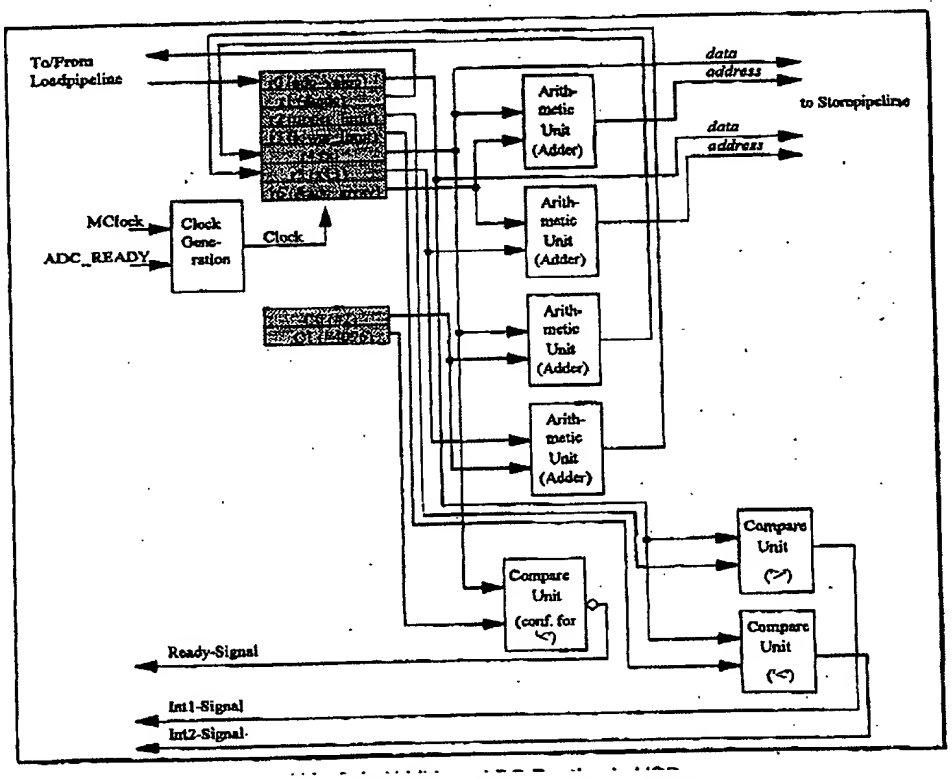

Fig. 2-4: Mapping ADC routine in UCB

This example shows the fundamental transferability of routines for intelligent peripherals to the concept of UCBs, where a number of assumptions had to be made in particular for the usability of procedural compilers. In particular, the system developer must declare the parts which are in each case intended to run in a UCB as independent threads (in the C example with the aid of the new keyword hardware_thread), while the compiler itself is required to translate this for a UCB.

2.4 Summary

The aim of this concept was to introduce a new architecture for field programmable logic which can simultaneously be integrated in microcontrollers and microprocessors and can be loaded by means of a sequential instruction stream with the aid of the PDSP algorithm (Procedural Driven Structural Programming) already specified. This architecture is distinguished by the introduction of a newly structured hardware optimized for the scope of application.

By way of example, it was possible to show in this case the form in which the UCBs can be programmed and operated. This structure can be operated both in standalone fashion and in conjunction with microcontrollers, and produces very good results, especially in the context of a co-design, by close matching to the architecture and method of operation of processors.

3 References

[1] De Micheli, G.; Gupta, R., "Hardware/Software Co-Design", Invited Paper in Proceedings of the IEEE Vol. 85(3), Special Issue on Hardware/Software Co-Design, 341 ... 365, March 1997.

[2] Siemers, C.; Prozessor mit Pipelining-Aufbau [Processor with a Pipelining Structure]. Application dated August 21, 1996 and numbered 196 34 031.4 to the German Patent Office.

[3] Siemers, C.; Möller, D.P.F., "Der >S<puter: Ein dynamisch rekonfigurierbares Mikroarchitekturmodell zur Erreichung des maximalen Instruktionsparallelitätsgrades [The >S<puter: A dynamically reconfigurable microarchitecture model for achieving the maximum level of instruction parallelity]", Proceedings of the 14th ITG/GI Symposium of Architecture of Computation Systems ARCS '97, Rostock, September 1997, pages 133 ... 142. VDE Verlag, Berlin and Offenbach, 1997.

[4] Siemer, C.; Möller, D.P.F., "The >S<puter: A Novel Microarchitecturemodel for the Execution of Instructions inside Processors", in Xu De, K.-E. Großpietsch, Ch. Steigner (Eds.): Proceedings of the Second Sino-German Workshop on Advanced Parallel Processing Technologies APPT '97, Coblenz, September 1997, p. 75 ... 82. Fölbach Verlag, Coblenz, 1997.

[4] Smith, James E.; Sohi, Gurindar S., "The Microarchitecture of Superscalar Processors", Invited Paper in Proceedings of the IEEE, Special Issue on Microprocessors, Vol. **83** (12), p. 1609 ... 1624, 1995.

[5] Wen-Mei W. Hwu et al., "Compiler Technology for Future Microprocessors", Invited Paper in Proceedings of the IEEE Vol. **83** (12), Special Issue on Microprocessors, 1625 ... 1640, Dec. 1995.

[6] R.W. Hartenstein, J. Becker, R. Kress, "Custom Computer Machines vs. Hardware/ Software Co-Design: From a Globalized Point of View", *6th Int. Workshop on Field Programmable Logic and Appl., FPL '96,* Darmstadt, Sept. 1996, Lecture Notes in Computer Science 1142, Springer 1996.